

Rapport Projet : Métro c'est trop

Anis BOUZIANE Julien JACQUET Damien DEMONTIS

5 mai 2017

Note : La methode utilisée pour écrire le code ne permet pas d'afficher les accents et les lignes trop longues sont coupées, cependant l'intégralité du code est également disponible a l'adresse suivante : <https://github.com/SolowinFXVI/Metrocesttrop>

1 Explication des structures de données :

La premiere structure *SOMMET* sert à stocker toutes les informations relative a un sommet/station (index du sommet,nom de la station, ligne de métro, status de terminus ou non). Pour stocker ces informations on va utiliser des tableau de *char* avec des tailles prédéfinies, cette technique consomme plus de mémoire que nécessaire mais évite l'utilisation de *mallocs*.

La seconde structure *TAB* stock les informations relatives a tous les sommets/stations. C'est un tableau de la structure précédente.

La troisième structure *ARC* contient les informations relatives a l'existence d'un arc c'est a dire un sommet de depart de type *sommet*, un sommet d'arrivée de type *sommet* également, ainsi que le poids de cet arc ici défini par le temps de trajet entre ces deux sommets.

La quatrième structure *G* est la matrice d'adjascence du graph elle est de type *ARC* et contient par conséquent toutes les informations des structures précédentes.

2 Principales procédures :

La principale procédure est la fonction *plus_court_chemin* sans aucun doute mais c'est au coeur de cette fonction que se trouve la partie la plus importante du programme : la partie *dijkstra*. Pour résoudre notre problème de plus court chemin nous avons décidé d'implémenter l'algorithme de dijkstra. Tout d'abord parce que c'est celui que nous avons vu en cours mais aussi parce que c'est un algorithme réputé. Il existe d'autres solutions au probleme du plus court chemin mais dijkstra semblait l'algorithme le plus adapté.

Il serait intéressant de savoir quel algorithme est utilisé par l'application de la sncf/ratp.Est-ce l'algorithme de dijkstra? Gourmand mais qui donne dans tous les cas le plus court chemin. Ou un algorithme de type A*, moins précis mais plus rapide?

3 Travail effectué :

Pour mener a bien ce projet il a fallut modifier grandement le fichier metro.txt, de facon a obtenir un formatage pratique à utiliser. Nous avons eu des difficultés avec la partie acquisition des données et finalement le programme est fonctionnel mais sans affichage graphique.

4 CODE COMPLET :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "lecture.h"
5 #include "mct.h"
6 #include "dijkstra.h"
7
8 TAB initialise_sommets(char *str, TAB M){
9     printf("initialise_sommets\n");
10    M=init_s(M);
11    return initialiser_sommets(str, M);
12 }
13
14 int conversion_sommet_str_sommet_int(TAB M, char
    *sommet_str){
15    printf("conversion_sommet_str_sommet_int\n");
16    int sommet=-1;
17    int i;
18
19    for(i=0; i<NBR_STATIONS;i++){
20        if(strcmp(sommet_str,M.TAB[i].nom)==0){
21            sommet = atoi(M.TAB[i].index);
22            return sommet;
23        }
24    }
25    if(sommet == -1){
26        printf("sommet inconnu ou mal orthographi \n");
27        printf("ex1: Basilique de Saint-Denis ->
            Basilique_de_Saint-Denis\n");
28        printf("ex2: Place d'Italie -> Place_d'Italie\n");
29        exit(EXIT_FAILURE);
30    }
31    return sommet;
32 }
33
34 //////////////////////////////////////////////////
35 int main(int argc, char *argv[]) {
36     TAB M;
37     char *sommet_depart_str;
38     char *sommet_arrivee_str;
39
40     if(argc < 3){
41         printf("Trop peut d'arguments\n");
42         printf("utilisation :\n");
43         printf("mct station_de_depart station_d'arrivee\n");
44         exit(EXIT_FAILURE);
45     }
46
47     if(argc == 3){
48         sommet_depart_str=argv[1];
49         sommet_arrivee_str=argv[2];
```

```

50 }
51
52 if(argc>3){
53     printf("Trop d'arguments\n");
54     printf("utilisation :\n");
55     printf("mct station_de_depart station_d'arrivee\n");
56     exit(EXIT_FAILURE);
57 }
58
59 M=initialise_sommets("metro.txt",M);
60 printf("sommets M initialis s\n");
61 initialise_graph("metro.txt",G,M);
62 printf("graph initialis \n");
63 plus_court_chemin(G,M,conversion_sommet_str_sommet_int(M,sommet_depart_str),conve
64 printf("fin algorithme\n");
65 printf("showme: %s %s %s\n",
        G[238][0].sm1.nom,G[238][0].sm2.nom,G[238][0].temps);
66 return 0;
67 }
68 ///////////////////////////////////////////////////////////////////
        ../src/mct.c

```

```
1  #ifndef __MCT_H
2  #define __MCT_H
3
4  #include "constantes.h"
5  #include "structures.h"
6  #include "lecture.h"
7  //STAT initialiser_sommets_et_arcs(char *str);
8
9  void calcul_trajet();
10
11 void afficher_trajet();
12
13 #endif
```

../src/mct.h

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "lecture.h"
5
6  TAB init_s(TAB M){ //remplie le tableau des sommet d'un
    marqueur "UNKNOWN"
7      int i;
8      for(i=0; i<NBR_STATIONS-1; i++){
9          strcpy(M.TAB[i].index,"UNKNOWN");
10         strcpy(M.TAB[i].nom,"UNKNOWN");
11         strcpy(M.TAB[i].ligne,"UNKNOWN");
12         strcpy(M.TAB[i].status,"UNKNOWN");
13     }
14     return M;
15 }
16
17 void init_g(ARC G[NBR_ARCS][NBR_ARCS]){//remplie la
    structure G de marqueurs "UNKNOWN"
18     int i;
19     int j;
20     for(i=0;i<NBR_ARCS;i++){
21         for(j=0;j<NBR_ARCS;j++){
22             strcpy(G[i][j].sm1.index,"UNKNOWN");
23             strcpy(G[i][j].sm1.nom,"UNKNOWN");
24             strcpy(G[i][j].sm1.ligne,"UNKNOWN");
25             strcpy(G[i][j].sm1.status,"UNKNOWN");
26             strcpy(G[i][j].sm2.index,"UNKNOWN");
27             strcpy(G[i][j].sm2.nom,"UNKNOWN");
28             strcpy(G[i][j].sm2.ligne,"UNKNOWN");
29             strcpy(G[i][j].sm2.status,"UNKNOWN");
30             strcpy(G[i][j].temps,"99999");
31         }
32     }
33 }
34 }
35
36 TAB initialiser_sommets(char *str, TAB M){ //extraire les
    donn es de "metro.txt" et les stock dans le tableau des
    sommets
37     int i;
38     char type[128];
39     char index[128];
40     char nom[128];
41     char ligne[128];
42     char status[128];
43     FILE* fic = fopen(str, "r");
44
45     if (fic == NULL) {
46         printf("echec ouverture fichier%s\n", str);
47         exit(EXIT_FAILURE);
48     }
49
50     for(i=0;i<NBR_STATIONS;i++){ //ignore ce qu'il y a apr s

```

```

        la derniere station
51     fscanf(fic,"%s %s %s %s %s\n", &type[0], &index[0],
        &nom[0], &ligne[0], &status[0]);
52     strcpy(M.TAB[i].index,index);
53     strcpy(M.TAB[i].nom,nom);
54     strcpy(M.TAB[i].ligne,ligne);
55     strcpy(M.TAB[i].status,status);
56 }
57
58 fclose(fic);
59 return M;
60 }
61
62 void associer_graph_data(char *sm1, char *sm2,ARC
    G[NBR_ARCS][NBR_ARCS],TAB M){
63     //associe les donn es stock es dans le tableau des
        sommets a la structure G
64     int i;
65     for(i=0;i<NBR_STATIONS;i++){
66         if(atoi(sm1) == atoi(M.TAB[i].index)){
67             strcpy(G[atoi(sm1)][atoi(sm2)].sm1.index,M.TAB[i].index);
68             strcpy(G[atoi(sm1)][atoi(sm2)].sm1.nom,M.TAB[i].nom);
69             strcpy(G[atoi(sm1)][atoi(sm2)].sm1.ligne,M.TAB[i].ligne);
70             strcpy(G[atoi(sm1)][atoi(sm2)].sm1.status,M.TAB[i].status);
71         }
72         if(atoi(sm2) == atoi(M.TAB[i].index)){
73             strcpy(G[atoi(sm1)][atoi(sm2)].sm2.index,M.TAB[i].index);
74             strcpy(G[atoi(sm1)][atoi(sm2)].sm2.nom,M.TAB[i].nom);
75             strcpy(G[atoi(sm1)][atoi(sm2)].sm2.ligne,M.TAB[i].ligne);
76             strcpy(G[atoi(sm1)][atoi(sm2)].sm2.status,M.TAB[i].status);
77         }
78     }
79 }
80
81 void initialiser_graph(char *str,ARC G[NBR_ARCS][NBR_ARCS],
    TAB M){ //extrait les donn es de "metro.txt" et les
        stock dans le tableau des arcs
82     int i=0;
83     char type[128];
84     char sm1[128];
85     char sm2[128];
86     char temps[128];
87     char garbage[128];
88     FILE* fic = fopen(str, "r");
89
90     if (fic == NULL) {
91         printf("echec ouverture fichier%s\n", str);
92         exit(EXIT_FAILURE);
93     }
94
95     while(fscanf(fic,"%s %s %s %s %s\n", &type[0], &sm1[0],
        &sm2[0], &temps[0], &garbage[0]) != EOF){
96         if(i>=376){ //ignore toute la partie sur les sommets
97             associer_graph_data(sm1,sm2,G,M);

```

```

98         associer_graph_data(sm2,sm1,G,M);
99         strcpy(G[atoi(sm1)][atoi(sm2)].temps,temps);
100        strcpy(G[atoi(sm2)][atoi(sm1)].temps,temps);
101        i++;
102    }
103    i++;
104 }
105 fclose(fic);
106 }
107
108 void initialise_graph(char *str,ARC
    G[NBR_ARCS][NBR_ARCS],TAB M){ //fonction
    d'initialisation complete de la structure G
109 printf("initialise_graph\n");
110 init_g(G);
111 initialiser_graph(str,G,M);
112 }

```

../src/lecture.c


```

1  #ifndef __LECTURE_H
2  #define __LECTURE_H
3
4  #include "constantes.h"
5  #include "structures.h"
6
7  TAB init_s(TAB M);
8  TAB initialiser_sommets(char *str, TAB M);
9  TAB cleanup(TAB M);
10
11 void initialise_graph(char *str, ARC G[NBR_ARCS][NBR_ARCS],
    TAB M);
12 void init_g(ARC G[NBR_ARCS][NBR_ARCS]);
13
14 #endif

```

../src/lecture.h

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "constantes.h"
4  #include "structures.h"
5  #include "string.h"
6
7  List fill_start(List l, int a){ //implementation classique
    d'une liste
8      printf("fill_start\n");
9      Element* new= malloc(sizeof(Element));
10     new->val = a;
11     new->next=l;
12     l=new;
13     return l;
14 }
15
16 void print_list_path(List l, TAB M,ARC
    G[NBR_ARCS][NBR_ARCS],int temps_total){//affiche la liste
17 if(l==NULL || l->next == NULL){
18     printf("liste vide\n");
19     return ;
20 }
21 while(l->next != NULL){
22     printf("De %s allez jusqu'a %s ligne
        %s\n",M.TAB[l->val].nom, M.TAB[l->next->val].nom,
        G[l->val][l->next->val].sm2.ligne);
23     l=l->next;
24 }
25 }
26
27 int conversion_temps_min(int temps_total){ //simple
    conversion du temps en minutes
28 int minutes;
29 minutes=temps_total/60;
30 return minutes;
31 }
32
33 int conversion_temps_sec(int temps_total){//va de paire
    avec la fonction conversion_temps_min et renvoie les
    secondes restantes
34 int secondes;
35 secondes=temps_total%60;
36 return secondes;
37 }
38
39
40
41 int sommets_tous_traites(int
    sommets_traites[NBR_STATIONS]){//test si le tableau ne
    contient que des 1 -> retourne 1
42 int i;
43 for(i=0; i<NBR_STATIONS; i++){
44     if(sommets_traites[i]==0){
45         return 0;

```

```

46     }
47 }
48 return 1;
49 }
50
51 void init_pere(int pere[NBR_ARCS]){//initialise le tableau
    pere a -1
52 printf("init_pere \n");
53 int i;
54 for(i=0; i<NBR_STATIONS; i++){
55     pere[i]=-1;
56 }
57 }
58
59
60 void dijkstra(int pere[NBR_STATIONS], int
    sommet_depart){//Transforme le tabelau pere en tableau
    des plus courts chemins
61 printf("dijkstra \n");
62
63 int sommets_traites[NBR_STATIONS]={0};
64 int plus_courte_distance[NBR_STATIONS];
65 sommets_traites[sommet_depart] = 1;
66 int i;
67
68 init_pere(pere);
69
70 for(i=0; i < NBR_STATIONS; i++){ //initialisation du
    tableau plus courte distance
71     if(atoi(G[sommet_depart][i].temps) != INFINI){//si i
        successeur du sommet de depart
72         plus_courte_distance[i]=atoi(G[sommet_depart][i].temps);//alors
            plus courte distance vaut le poids de l'arc
73         pere[i]= sommet_depart;
74     }
75     else{ //sinon plus courte distance vaut l'infini
76         plus_courte_distance[i]=INFINI;
77     }
78 }
79
80 plus_courte_distance[sommet_depart] = 0;
81
82 //////////////////////////////////////
83 while(!sommets_tous_traites(sommets_traites)){
84     printf("tours de while\n");
85     //recherche du prochain sommet
86     int min = INFINI, a_traiter;
87
88     for(i=0; i<NBR_STATIONS; i++){//pour toutes les stations
89
90         if(sommets_traites[i] == 0){//si le sommet n'est pas
            trait
91             if(plus_courte_distance[i] <= min){//si la plus
                courte distance est inferieure au minimum actuel

```

```

92         a_traiter = i;
93         min=plus_courte_distance[i];
94     }
95 }
96 }
97
98 sommets_traites[a_traiter] = 1;
99
100 for(i=0; i<NBR_STATIONS;i++){//pour les successeurs du
    sommet a_traiter
101     if(atoi(G[a_traiter][i].temps) != INFINI){//test si
        la distance de ce sommet au sommet de depart est
        plus grande que celle passant par le sommet
        a_traiter
102     if(plus_courte_distance[i] >=
        plus_courte_distance[a_traiter] +
        atoi(G[a_traiter][i].temps)){
103         //mise a jour de la distance
104         plus_courte_distance[i] =
            plus_courte_distance[a_traiter] +
            atoi(G[a_traiter][i].temps);
105         //donne le pere
106         pere[i] = a_traiter;
107     }
108 }
109 }
110 }
111 ///////////////////////////////////////////////////
112 }
113
114 void plus_court_chemin(ARC G[NBR_ARCS][NBR_ARCS],TAB M, int
    sommet_depart, int sommet_arrivee){//calcul le plus
    court chemin entre deux sommets
115 printf(" plus_court_chemin\n");
116 if(sommet_depart == sommet_arrivee){
117     printf("Vous etes deja arriv ...");
118     return ;
119 }
120
121 int pere[NBR_ARCS]; //tableau utilis par dijkstra
122 List path = NULL;//liste de stockage du plus court chemin
123 int temps_total = 0;
124
125 dijkstra(pere,sommet_depart); //COEUR DU PROGRAMME :
    modifie le tableau pere
126
127 int depart, arrivee = sommet_arrivee;
128
129 path = fill_start(path,arrivee);//remplissage du debut de
    la liste
130
131 do {
132     depart = pere[arrivee]; //sommet de depart actuel est
        le pere du sommet d'arriv e

```

```

133     if(depart == -1){
134         printf("Pas de chemin possible entre %s et
           %s\n",M.TAB[sommet_depart].nom,
           M.TAB[sommet_arrivee].nom);
135         return ;
136     }
137     temps_total += atoi(G[depart][arrivee].temps);
           //incr mentation du temps
138     path=fill_start(path,depart);//remplissage de la liste
139     arrivee=depart;
140 }while(depart != sommet_depart);
141 printf("\n");
142 print_list_path(path,M,G,temps_total);//affichage du
           trajet
143 printf("\n");
144 printf("temps de trajet : %d minutes et %d
           secondes\n",conversion_temps_min(temps_total),conversion_temps_sec(temps_total)
145 //affichage du temps de trajet
146 printf("\n");
147 }

```

../src/dijkstra.c

```

1  #ifndef __DIJKSTRA_H
2  #define __DIJKSTRA_H
3
4  #include "structures.h"
5  void print_list_path(List path, TAB M, ARC
        G[NBR_ARCS][NBR_ARCS], int temps_total);
6  void dijkstra(int pere[NBR_STATIONS], int sommet_depart);
7
8  void plus_court_chemin(ARC G[NBR_ARCS][NBR_ARCS], TAB M, int
        sommet_depart, int sommet_arrivee);
9
10 #endif

```

../src/dijkstra.h

```

1  #ifndef __MCT_H
2  #define __MCT_H
3
4  struct sommet{ //structure contenant les informations
                    relatives a un sommet
5      char index[128];
6      char nom[128];
7      char ligne[128];
8      char status[128];
9  };
10 typedef struct sommet SOMMET;
11
12 struct tab{//structure contenant les informations relatives
                    a tous les sommets
13     struct sommet TAB[377];
14 };
15 typedef struct tab TAB;
16
17 struct arc{//structure contenant les informations relatives
                    a l'existence d'un arc
18     struct sommet sm1;
19     struct sommet sm2;
20     char temps[128];
21 };
22 typedef struct arc ARC;
23
24 ARC G[472][472]; //Matrice d'adjascence utilis e par la
                    partie dijkstra
25
26 struct element{ //liste pour stocker le trajet
27     int val;
28     struct element* next;
29 };
30
31 typedef struct element Element;
32 typedef Element* List;
33
34 #endif

```

../src/structures.h

```
1 #define NBR_STATIONS 376
2 #define NBR_ARCS 472
3 #define TAILLE_NOM 200
4 #define TAILLE_LIGNE 5
5 #define INFINI 99999
    ../src/constantes.h
```



```

1 CFLAGS=-c -g -Wall
2
3 all: mct
4     ./mct
5
6 compil: mct
7     make mct
8
9 test: mct.o lecture.o dijkstra.o
10    make clean
11    make mct.o
12    make lecture.o
13    make dijkstra.o
14    gcc -o tes mct.o lecture.o dijkstra.o
15    ./tes Arts_et_M tiers Galli ni
16
17 mct.o: mct.c mct.h constantes.h lecture.h structures.h
18        dijkstra.c dijkstra.h
19    gcc $(CFLAGS) mct.c
20
21 lecture.o: mct.c mct.h constantes.h lecture.h
22        structures.h dijkstra.c dijkstra.h
23    gcc $(CFLAGS) lecture.c
24
25 dijkstra.o: mct.c mct.h constantes.h lecture.h structures.h
26        dijkstra.c dijkstra.h
27    gcc $(CFLAGS) dijkstra.c
28
29 clean:
30    rm -f mct
31    rm -f mct.o
32    rm -f lecture.o
33    rm -f dijkstra.o

```

../src/Makefile